

WISE Expert: An Expert System for Monitoring Ship Cargo Handling

T. R. Addis

University of Portsmouth

Tom.Addis@port.ac.uk

www.tech.port.ac.uk/staff/addist/intel.htm

J. J. Townsend Addis

Clarity Support Ltd

Jan.Addis@clarity-support.com

www.clarity-support.com

R. Gillett

L3 MPRI Ship Analytics

pchemx@interalpha.co.uk

http://www.shipanalytics.com/MS/wise_index.asp

Abstract

WISE Expert is a general-purpose system that can be used for monitoring or controlling, in real time, complex systems that have recurring sub-structures. The system has been developed using a unique schematic development tool that ensures coherency of structure during design and construction. The design of the Expert System takes advantage of a distinction between the monitored system structure and expert knowledge so that the structure description can be used to generate specific rules for the system automatically. The system has been tested as an overseer during the running of trainee mariner exercises with a liquid cargo simulator and is now operational at over **35** customer sites throughout the world.

1 Introduction

WISE Expert is an expert system that was initially designed specifically to monitor a simulation of liquid cargo vessels.

As part of the process of training ships officers simulators are used to provide some of the training in a real world environment. With respect to the handling of bulk liquid cargoes such as chemicals, liquefied gases and oil a Liquid Cargo Handling Simulator (Lchs) is used. Such a simulator provides a representation of all the cargo storage tanks and associated pipeline systems, valves, control systems and equipment that are required to load/unload the cargo and maintain it in an appropriate condition during the voyage. The models simulate all the thermo and fluid dynamics of the various mixtures of liquids and gases within the systems and

hence together allow students to undertake all the tasks associated with preparing the vessel and handling the appropriate cargoes. Traditionally simulators of this type were shore based, located in specific training institutions. This was because of the need for both the hardware and skilled instructors (or overseers). They were required to provide the training and monitor the student's activities during the simulation exercise to ensure that the student did not learn bad techniques, pursue bad practices or enter into dangerous situations during cargo management exercises. With the introduction of PC technology, the hardware became less of an issue but the need for a skilled instructor remained. In 1997 it was becoming apparent that with the reduction in crew numbers the shipping companies were experiencing increasing logistical difficulties in being able to release sufficient personnel to attend shore based courses. Consequently, to ensure the training in this field could be delivered at the required level it was decided to investigate a way in which the need for the skilled instructor could be replaced by a computer based system. This allows the simulator to be used safely by a student for self-study without an instructor being available, such as in an onboard environment [1].

The Lchs cannot only be used to simulate different vessel types but also different variations of the same type, such as oil tankers of varying capacities. In the latter case the basic structure of the vessel and the principles of operation remain the same but the actual configuration of pumps, tanks and other structural elements will change. Consequently, it was important in designing WISE Expert to ensure that the knowledge base produced for one vessel could be easily adapted for use on another vessel of the same type but of a different configuration. This would then allow new models to be produced quickly and easily.

It was proposed in 1998 that an expert system could be created that would take the place of the marine tutor in monitoring students' actions and warning students of potential or real hazards during exercises.

If such a system could be created then this would lead to a range of other uses such as improved classroom response and the monitoring of real ships at sea. Improved classroom response could be achieved if student monitoring could also be networked. Then major warnings could be fed back to a central overseeing station. This would then release the tutor to concentrate on difficult problems and thus improve the student-learning environment. The monitoring of real ships at sea, if logged, would also provide feedback into the expert system from actual events. This would produce a continually improved product, providing a richer resource for training and extending the capability of a shipboard monitor.

Investigations of a sample of expert system products showed no commercial off-the-shelf systems that can cope with this requirement [2]. The problems with these standard expert systems were:

- Expensive – commercial licensing
- Self contained – links to the simulator would be difficult
- Networking – not available.
- Knowledge only in rule form – ship structures for example would have to be defined as rules (see section 2)
- No assurance that there is coherence of the knowledge base

- Presumptions of use – we are not requiring a ‘solution’ to be discovered. We have no goal other than monitoring for problems (although control goals are also a potential option).
- Limited extendibility – decisions based on calculations are restricted
- Slow – all rules work through a general inference engine
- Depth first assumption – only crude heuristic control over rule access order is possible.
- There seemed no way to take advantage of the repetitiveness of ship storage systems for automatic rule generation thus saving the expert effort.

It was decided to create a special expert system from first principles that would address the unique problem of overseeing the simulator. It had to have the following properties:

- ✓ Inexpensive – no license fee and minimum development time
- ✓ Easy links to other programs and threads
- ✓ Networked communication between different simulations
- ✓ Communication hub for an overseeing instructor
- ✓ Pattern sensitive monitoring rather than goal seeking.
- ✓ Extendable – so that any kind of procedure can be done within a decision rule
- ✓ Flexible – be able to change easily for different ships, addition of new rules to be applicable to all ships. For example, the automatic generation of rules from structural information
- ✓ Continuous development of the rule base to be rendered in house
- ✓ Knowledge base to be limited to only explicitly specified dependencies
- ✓ Fast – a complete scan of all possibilities should take no more than 10 seconds
- ✓ Control over rule access to be under precise control

Further, given the right development tool it is not too difficult to create a bespoke expert system [2]. We can then have complete control over our design by avoiding the constraints of someone else’s paradigm. The first operational system was designed and constructed in about four man-months.

2 Rules for all Occasions – The Standard Approach

One of the obvious characteristics of a liquid cargo ship is that it consists of a collection of tanks. Each tank represents a single system that has valves and gauges indicating its state. There will be rules associated with each tank such as:

Rule#	Monitoring Rules	Message/Meaning/Action
1	If Cargo tank pressure < 0 & Cargo tank P/R valve open	Cargo tank Object (Rule#1) in vacuum. Relief valve open
2	If Cargo tank pressure >= 1400 & Cargo tank P/R valve open	Very high pressure in cargo tank Object (Rule#2)

If this were to be done by a typical expert system then associated with these rules would be another set of rules that define the structure of the ship. Here rules include facts and clauses [2]. The need for structure rules in the expert system is that the conditions to be monitored depend upon a different but overlapping set of structure relationships than used by the simulator. These expert structure rules would include the 3000 different variables that define the state of a simulation (or

even a real ship with respect to the cargo). Other rules would be there to identify the different objects (e.g. tanks). There would also be rules that simply act as intermediate links between other rules to allow inference to occur. For a typical liquid cargo vessel we estimate that there would be about 10,000 or more rules of this kind. The process of automating the overseeing would be to take each Monitoring Rule and expand it using the Structure Rules.

Rule#	Structure Rules	Variable/Object/Relation
S1	If Cargo tank pressure	Variable VPP(1)
S2	If Cargo tank pressure	Variable VPP(2)
O1	If VPP(1)	Object (Number 1, Port)
O2	If VPP(2)	Object (Number 1, Starboard)
S17	If Cargo tank P/R valve	Variable FLOPAR(? ,303)
S18	If Cargo tank P/R valve	Variable FLOPAR(? ,304)
O17	If FLOPAR(? ,303)	Object (Number 1, Port)
O18	If FLOPAR(? ,304)	Object (Number 1, Starboard)

For example in Monitoring Rule#1 above we would need to take all the alternatives that the structure rules express. In a typical case this would be replaced by 16 or so possibilities and would require at least three stages of inference for each of them not including all the potential paths that fail.

Rule#	Constructed Rules	Constructed Message/Meaning/Action
C1	If VPP(1) < 0 & FLOPAR(? ,303) = Open	Cargo tank (Number 1 & Port) in vacuum. Relief valve open
C2	If VPP(2) < 0 & FLOPAR(? ,304) = Open	Cargo tank (Number 1 & Starboard) in vacuum. Relief valve open

The cycle would involve for each variable referenced in the Monitoring Rule the finding of a Construct Rule that has an **If** component that matches the variable reference (e.g. Cargo tank pressure) and also has the same object reference as all the other variables in the Monitoring Rule (e.g. Number 1, Port). The object reference is then used in the Message as well as ensuring the correct binding between the different tests on the variables.

The collection of tanks also represents systems that interact with each other. An example of such an interaction is:

Rule#	Monitoring Rules	Message/Meaning/Action
3	If Rule#1 & Rule#2 apply to adjacent tanks	Large pressure differential between cargo tanks Object (Rule#1) and Object (Rule#2) . Possible conditions leading to bulkhead collapse.

This requires some further structural details that describe the adjacency of tanks. Other kinds of relationships can also be expressed (e.g. Connected_To).

Rule#	Structure Rules	Variable/Object/Relation
A1	If Object (Number 1, Port)	Connected_To (Object (Number 1, Starboard))
A2	If Object (Number 1, Port)	Connected_To (Object (Number 2, Port))
A3	If Object (Number 2, Port)	Connected_To (Object (Number 2, Starboard))
A4	If Object (Number 2, Port)	Connected_To (Object (Number 1, Port))
A5	If Object (Number 2, Port)	Connected_To (Object (Number 3, Port))

The cargo tanks are also supported by other systems such as the inert gas system that will have rules such as:

Rule#	Monitoring Rules	Message/Meaning/Action
4	If Inert Gas Power fail & {Scrubber pump is running or Seal pump is running or Fan is running}	Pumps cannot be started. Power is not available.

Overseeing requires an exhaustive scan of all the Monitoring Rules because it is looking for patterns that need a response from the students or leads to such a response and the expansion of these rules using the structure rules has to be achieved every 10 seconds. We expect that for a typical ship there would be about 2000 or more monitoring rules of this kind coupled with the 10,000 or more structural rules. At the very best this means some 6000-inference steps (not counting failures) in the time limit of 10 seconds. This requires each inference step to be completed in less than 0.0017 second or about 250 to 500 machine cycles with current processor power (0.6 to 1.2 GHz Machines). The estimated time would take about 1.5 times longer than this best required time. This is possible, but it would have to be handcrafted and all the tests would have to be simple. There would be no time for the simulator (or anything else) to run on the same machine.

3 A Division of Labour

Since the structure of a ship does not normally change during overseeing then much of the inference required for rule construction can be done off-line. Consequently, the structural linking with the Monitoring Rules is pre-compiled by using a mechanism for rule generation from the ship's structure. This takes advantage of the repetition of the cargo organisation. A ship's structure can be expressed as a hierarchy (the context tree). This is not always the case but exceptions can easily fit into such a scheme. The rules that reference a class of ship in terms of the desired expertise are expressed as General Rules. The particular ship drawn from that class for which we need Specific Rules that describe the structure is called the Target System. The associated context tree is the Target System description and is used by a rule generator that will construct the rules for a specific ship. These resulting rules are the System Specific Rules (Figure 1).

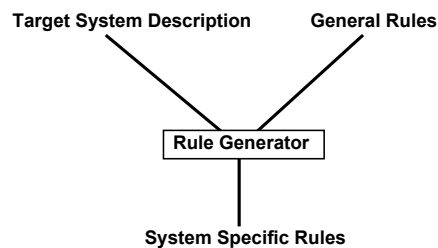


Figure 1. The Merging of Information for Rule Generation

Figure 2 shows a small part of a simplified context tree that represents the structure of a ship. The context tree takes the place of the structural rules that would have

been needed for the standard expert system. The root of the tree is SHIP and can always be assumed. The nodes along a particular pathway can be used as object (concept) descriptors as used in the FLIN conceptual database [2] (adjectives and nouns). So in our example rules rewritten below, CARGO TANK indicates the general type of object to which the rules refer. Alternatives might be CARGO PIPE or BALLAST TANK and so on.

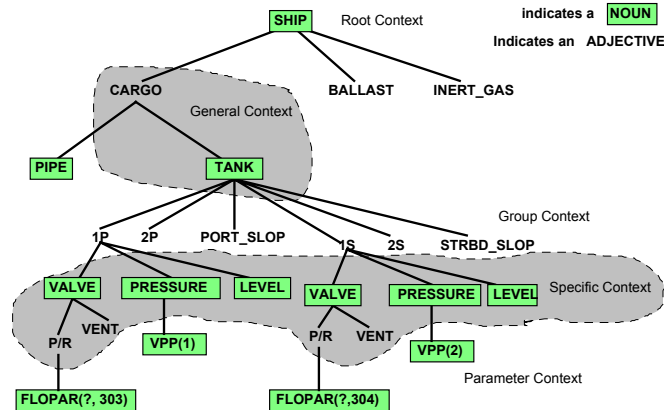


Figure 2. The Context Tree

Rule #	Object Type (General Context)	Monitoring Rules (Specific Context)	Message/Meaning/Action (Group Context)
1	Cargo tank	If pressure < 0 & P/R valve open	Cargo tank Object (Rule#1) in vacuum. Relief valve open
2	Cargo tank	If pressure >= 1400 & P/R valve open	Very high pressure in cargo tank Object (Rule#2)

The general object has parts associated with it. These parts can also be referenced in the same way. These specific objects like P/R VALVE or VENT VALVE or simply just PRESSURE depict general variables, which define the state of the object. Combine both the General and Specific Contexts then two things are identifiable. The first is the set of objects to which the rule applies and the second is the set of variables on each object that need to be tested. In this example the objects will be the set {1P, 2P, PORT_SLOP, 1S, 2S, STRBD_SLOP}. Note that the Group objects can consist of more than one node such as MAIN OXYGEN. Once these group items have been identified then the parameters (the variables in the simulation) can be named. All this can be done off-line to generate the constructed rules [2].

If it is found that more than one specific context variable of an object is identified during rule generation then that rule is applied to the 'or' of all its possibilities. For example, the pre-condition of Rule#1 might have been written (see figure 2):

For Cargo Tank If pressure < 0 and valve open

Then this would be interpreted, since there are two options, as:

For Cargo Tank If pressure < 0 and (P/R valve open or vent valve open)

If there is more than one ambiguous specific context variable then every possible combination (the cross product) is generated. This will be referred to as *the ambiguity construct*.

The hierarchy is stored as text in a file where each level is marked by a set of dashes as illustrated in table 1. Repeated structures are easy to reproduce through cut and paste.

```

SHIP
- CARGO
-- TANK
--- 1P
---- VALVE
----- P/R
----- FLOPAR(?..303)
----- VENT
----- FLOPAR(?..288)
--- PRESSURE
---- VPP(1)
-- 2P
  
```

Table 1. A Fragment of the structure hierarchy as stored on file

Relationships can also be represented in a context tree as shown in figure 3. Here the relationships are between objects that can be identified as single nodes. All the nodes in the context can be considered as relationships and as such the decomposition of the ship does not have to follow physical structures; they can include usage or functional relationships. For example, this may be the relationship of the liquid cargo group of pipes.

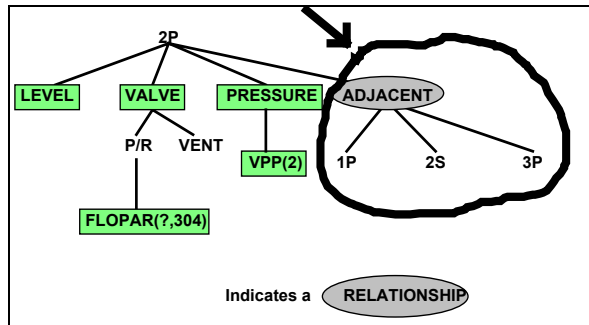


Figure 3. Relationships

4 Rules as Pictures

Since the development tool (ClarityPro) creates functional programs directly through diagrams, it was thus natural to create rules in the same form. Formal representation is essential for computation; nevertheless, formal notation may prematurely displace informal diagrammatic working during the process of developing a program or model [3]. It is conceivable both to be formal and informal at the same time because it is now possible to generate program code directly from

diagrams. These diagrams are the Clarity¹ schema and can be interpreted directly into the functional language [4, 5]. A typical rule schematic is shown in Figure 4. For reasons of rule management the single rule number has been replaced by three parameters that define the Main and Sub categories followed by a relative rule number within those categories. These three dimensions now define the identity of any rule. The user may pre-define the allowed categories. Internal to the system these three dimensions are converted into a single rule number.

Functions (and hence their schematic) normally have zero or more input parameters and always a single output. The inputs are at the top of the schematic and the output is at the bottom. However, such a schematic can also be considered to be similar to a completed form. The lozenges represent the form fields and these can be added to and/or changed so that a new rule can be created from an old rule.

Starting at the bottom of the schematic, the lozenge containing `<list ?0>` describes the type of output for the function `cnorm_&`. In this case it will be a list of mixed types (strictly *an unknown* type). This function `cnorm_&` is there to pre-process the rule into a form ready for the rule generation sequence. The user defines the general context of the rule in the next lozenge. In this case "CARGO" "TANK". The next cluster of lozenges expresses the pre-condition of the rule. This also includes all the specific contexts that help identify the variables to be tested and the additional logic connectors. These connectors include *and* (as `&&`), *not* and *or* (as `||`). The different pre-condition lozenges contain tests that are linked together via connecting arrows through these logic functions. The final cluster of lozenges that converge on the function `makelist` defines any number of the messages/meanings/actions of the rule. In this case, a Warning' is output, 'Keywords' is available for selection during rule management and 'Sw_On' marks the firing of this rule.

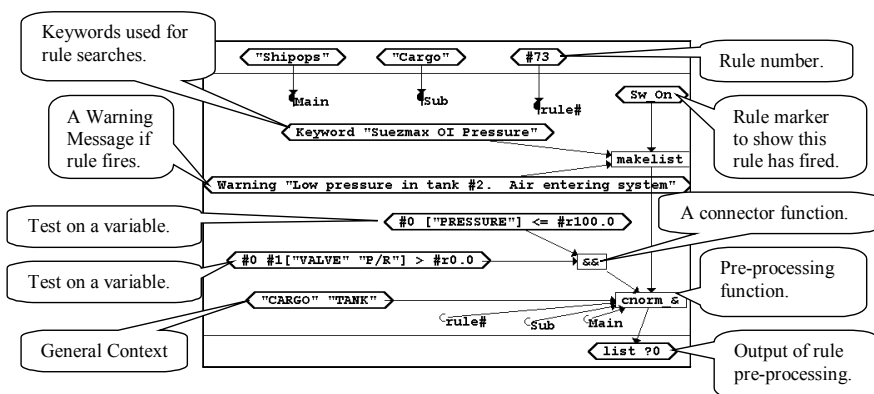


Figure 4. A Rule defined as a function as used in LICOS

There are several other kinds of consequences of the rule firing. This includes 'Action', 'Act', 'Information' and 'Error'. 'Act' and 'Action' provide a means of obeying functions that do something. Functions can also be obeyed in the pre-

¹ A version of Clarity (ClarityLite) minimised for teaching functional programming can be obtained free off the web: <http://www.clarity-support.com/>

condition provided they return a Boolean result. However, their side effects may be anything (e.g. starting a machine or printing). All processing begins at the bottom of the picture. The parameters are then processed clockwise from a function's output. The order of processing the pre-condition can be important.

5 Rule Pre-Conditions

The usual structure of a rule pre-condition is a set of tests linked together with logic connectors. These connectors are functions. Figure 5 is an example of a set of logic connectors. This schematic includes the pre-condition logic represented by:

```
(
    {Port Suction Valve > #r0}
    or {Stbd Suction Valve > #r0} )
and {Drive Valve > #r0}
and (not {Discharge Valve > #r0} )
```

Any formally correct Boolean structure may be constructed using these logic connector functions. The process of modifying the diagram is made simple through the ClarityPro environment.

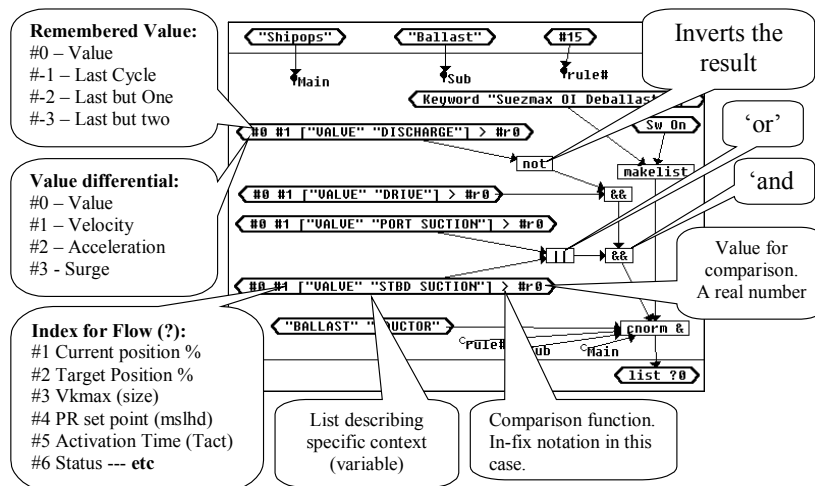


Figure 5. The structure of the pre-condition

The specification of the specific context variable can be extended to include three other elements:

- *Remembered Value*: The system remembers the last three values from previous cycles. The negative integer placed at the beginning of the test indicates which of the last three values is required for the test.
- *Value Differential*: The system uses these last three values to calculate the rates of change associated with a variable. Instead of a negative integer, a positive integer is used to indicate the level of differentiation. Most physical

systems can be described completely in terms of value (e.g. position), velocity (e.g. momentum), acceleration (e.g. force) and surge (e.g. impact).

- *Index to Flow*: The second integer refers only to certain variables in the simulator where there is an unspecified parameter. Such a parameter is marked by a '?' in the context tree. The system just replaces the '?' with the number. FLOPAR(?,303) is an example of a flow measurement that references a range of possible meanings depending in the value of '?'. Some examples are shown in figure 5.

Rule tests can be simply the fact that a specified rule has fired during the cycle. Figure 6 shows that all that's required is to give the number. The assumption is that the number is relative to the same main and sub category as the current rule. If the referenced rule is in a different category or sub-category then instead of the single number the following format is used:

<Ext "Shipops" "Ballast" #2>

The signs < > indicate the lozenge. Round brackets are used with functions that are to be interpreted at the point of rule generation. This is needed, for example, because the expert system assumes an internal system of rule numbering and this mechanism ensures that all the rules will be translated into this form straight away. Square brackets are used to defer evaluation until the point where the rule is to be interpreted in the expert system. Figure 7 is an example of three deferred functions used in place of tests.

Normally the notation for functions is pre-fix. However, the function *store#* used in two of them takes advantage of the special in-fix notation of specific rule tests. This means that all the mechanisms for value identification, differentiation and past cycles can be used. The function *store#* retrieves the specific context value and places it in the numbered store location. The result of this side effect is True if successful otherwise False.

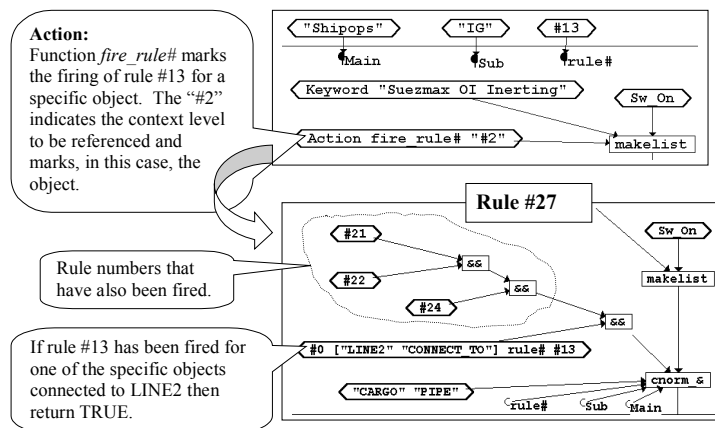


Figure 6. The passing of information between rules

The numeric reference to a memory location with *store#* is global so in principle could be used outside the rule. There is no mechanism here for marking the source of the information other than the number. The only limit to the number of locations is the limit set by the system for an integer.

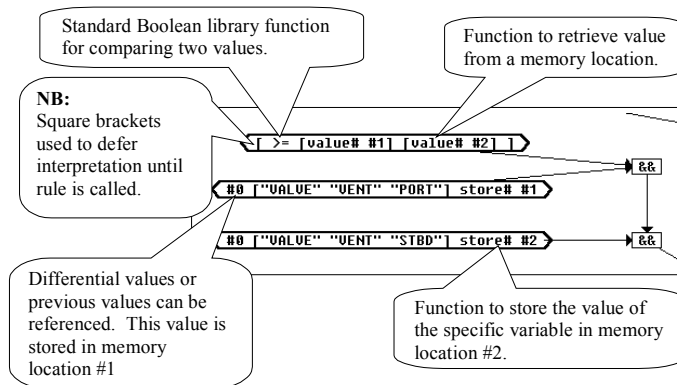


Figure 7. Using functions and memory in a Pre-condition

The final function in figure 7 uses the function *value#* to retrieve the stored values. This uses the normal pre-fix notation. Square brackets are used throughout in place of the normal round bracket so that all evaluations are deferred. Any library or user defined function can be used this way. There are currently about 300 library functions.

Rules can be used to trigger off any pattern directed event that can be programmed in ClarityPro. The limitation is that the actions must conform to the strict rule format. It is also possible to call the complete rule interpreter with a separate set of rules as a rule action. Thus, different monitoring procedures or expertise can be triggered by rules to any depth. However, generality is restricted because global variables such as switches and other inter-rule communication will remain current at all depths.

6 Cycles of Interpretation

The pre-compiled system specific rules are in two parts. The first part is the pre-condition and is in conjunctive normal form. The second part is a list of actions that are activated when the condition is True.

The pre-condition tests are ordered such that they are the conjunction of disjunctions (conjunctive normal form). Every Boolean function, which consists of Boolean tests that are linked together by normal logical connectives and brackets, can be converted into this normal form. An example of this is the condition that might be expressed as:

$$(a \wedge d) \vee (b \wedge c)$$

where 'a', 'b', etc are Boolean tests on variables. This will become in conjunctive normal form:

$$(a \vee b) \wedge (a \vee c) \wedge (d \vee b) \wedge (d \vee c)$$

The function 'Check Rule Boolean Condition' in figure 8 runs through each of the 'ored' conditions and if a True is detected within the 'ored' list testing will stop and return True otherwise False. At the next level up, the first False that is encountered in the 'anded' conditions will return False otherwise True. This ensures minimum testing of variables.

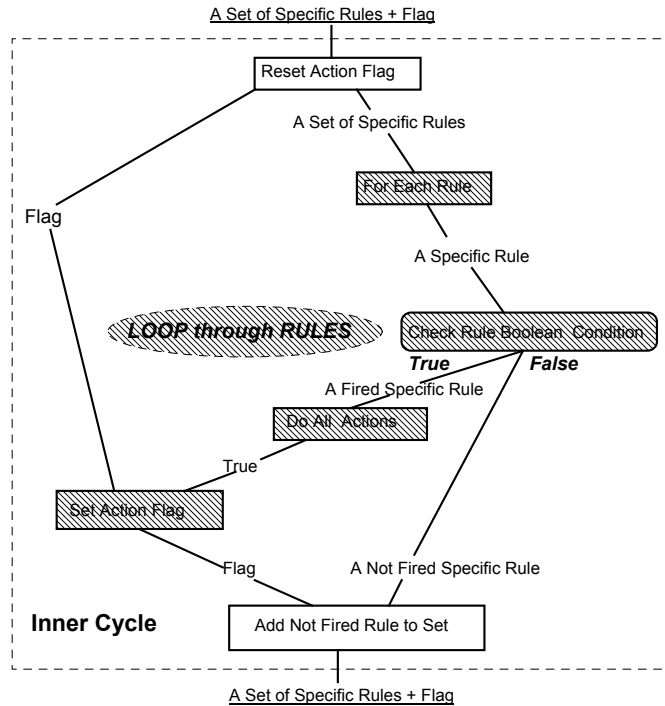


Figure 8. The Inner Cycle through a rule Set

Thus, for each rule in the set, if the pre-condition is True then the actions are obeyed and the rule is then effectively deleted from the set during this outer cycle (figure 9). All unfired rules where the pre-condition was False are collected together for reassessing. This is because the firing of a rule may alter the conditions of other rules so that they are then able to fire on the next round. For example, switches may be set by a fired rule that is tested by other rules. These dependent rule firings will occur in either the same inner cycle or the next one. The Action Flag is set to show that at least one rule has fired on that inner cycle.

The Outer Cycle ensures that the Inner Cycle is repeated until either there are no more rules left (they have all been fired) or no rules have been fired (nothing happened). At one of these two points the Outer Cycle finishes. This Outer Cycle is then repeated every N seconds where N is predefined by the set-up procedures.

This process ensures that the rules are *explored forward chaining and breadth first*. Since the structure has already been combined with the detailed rules the depth of the tree is likely to be shallow. The result is an exhaustive scan of the applicable rules every N seconds. In our case N has to be less than 10.

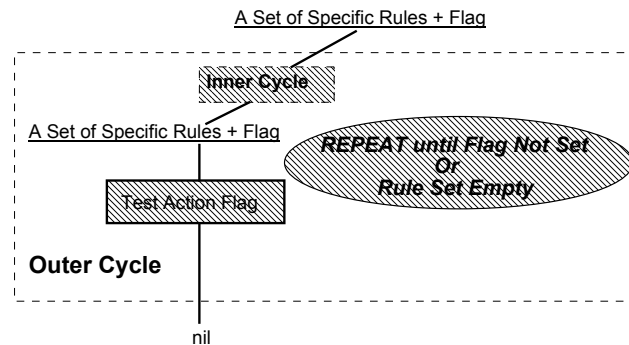


Figure 9. The Outer Cycle through the rule set.

Many processes, if kept simple to express, depend upon order. The mechanism of rule interpretation cannot guarantee that the rules will always be visited in a particular sequence. The only certainty that we have is that the processing within a rule will be consistent. What are also certain is that every rule will be visited at least once in an outer cycle and that every rule that fires will be eliminated on subsequent inner cycles. All switches etc. will be reset only after each complete outer cycle. It is possible in many cases to use this information to force sequencing between rules.

It is also known that any logical function can be expressed in conjunctive normal form. For example, we require firing a rule when all the stripping and suction valves attached to a particular line were **closed**. It is known that the ambiguity construct would 'or' all the valves together for any particular line but we need an 'and'. However, using De Morgan's Law we can get a simple logical equivalent by testing the valves for **open** and using an 'or'. So the statement:

All the valves are **closed** for line 1

becomes:

There are **no** valves **open** for line 1.

The system currently has about **2000** rules that are scanned in about **2** seconds. These rules are generated from a context tree of a ship and about **500** general rules.

7 Conclusion

The WISE Expert System, as part of the 'WISE Cargo' product has been purchased by 15+ companies (primarily ship owners and managers) and is operational in over **35** sites worldwide, including 25 installations on board real vessels, generating an income in excess of \$250,000.

There are two primary benefits identified by clients. First, is that by having the system on board it allows them to ensure all the appropriate officers obtain training on the simulator instead of just the few sent to shore based courses. Second, it allows them to either reduce the overall training costs (reduced travel) or to structure the training more effectively by using the higher cost shore based courses to concentrate on teaching the higher level skills and student assessment. The

system has also been very successful in providing small training centres with the ability to provide high-level simulator training without the traditional large overheads. The system is currently being extended to incorporate automatic assessment of the student's competence in undertaking a task. The assessment can be provided as feedback in real time to both student and instructor or as hardcopy for use by the clients. The system is also being adapted to provide the same capability for other types of simulator including those for Engine Room operations and also enhanced prediction and control for Emergency Management Systems.

The greatest advantage of the distinction between structural and expert knowledge has been the ability to use the captured expertise freely for new designs of ship as they arise and the distribution of new expertise gained for one ship across all other ship designs. The pre-compilation of this knowledge has also made the expert system responsive enough to be used in real time for operational cargo ships.

Ships officers very often only stay on a vessel for 4-6 months and do not return to the same vessel. Consequently, the 'knowledge' they build up about operating that particular vessel is lost when they leave and has to be learnt by their replacement, thus affecting both safety and efficiency. At the same time, there is widespread recognition that the overall standard of the skills of seafarers is reducing. With this in mind, work is currently in progress to connect WISE Expert with the control system of real ships instead of a simulation. In this context, the WISE Expert 'knowledge database' will provide a repository whereby the knowledge for the ship can be collected along with the general operational knowledge used with the simulator. WISE Expert will then be used to intelligently monitor all the activities onboard relating to the control of the cargo operation and provide warnings or advice to the operators if particular situations are identified or predicted, enhancing the overall safety capable of being provided by the ships operating system [1]. The first such system is expected to be completed during 2006.

References

1. Gillett, R. & Addis T. R., '[Intelligent Software: How it can be used to improve safety and training methods](#)'. GASTECH2000, The 19th International LNG, LPG & Natural Gas Conference & Exhibition, George R. Brown Convention Center, 14-17 November, Houston, Texas, USA. pp 10, (2000),
2. Addis T.R., '[Designing Knowledge Based Systems](#)'. published Kogan Page. ISBN 0 85038 859 7, ISBN 1 85091 251 3, 1985; Prentice Hall, ISBN 0-13-201823-3; 1986; 2nd Edition Ab-Libris Ltd., ISBN: 190356118 3 2003
3. Shaw L. G. & Woodward J. B '[Modeling Expert Knowledge](#)', Knowledge Acquisition ,vol. 2, pp. 179-206, (1990)
4. Addis T. R. and Townsend Addis J. J '[Avoiding Knotty Structures in Design: Schematic Functional Programming](#)', Journal of Visual Languages and Computing, Vol. 12. pp689-715. (2001)
5. Addis T. R. and Townsend Addis J. J. '[An Introduction to Clarity: A Schematic Functional Language for Managing the Design of Complex Systems](#)'. Vol. 56, No4, April, International Journal of Human Computer Studies, ISSN 1071 5819, pp331-422, (2002)